Analysis of an internet voting protocol

Kristian Gjøsteen

March 9, 2010

Abstract

The Norwegian government will trial internet voting in the 2011 local government elections. We describe and analyse a simplified version of the cryptographic protocol that will be used, and briefly describe the full version of the protocol.

This paper is a first step in a planned evaluation of the cryptographic protocol.

1 Introduction

The Norwegian government will trial internet voting in the 2011 local government elections. One of the key requirements for elections is trust, and for many internet voting deployments this trust has been lacking.

One reason is excessive secrecy. It is a well-established cryptographic and computer security principle that secrecy does not ensure security. Empirically, the converse appears to hold as transparency seems to increase security. We observe that making system architecture, design and even implementation details available for inspection leads to increased security, at least in the long run.

In order to build trust in internet voting, the Norwegian government has decided on nearly complete transparency. Most important documents from the tender process, including most technical details in every submitted proposal, has been made public. Before the 2011 trial, the architecture and even the source code for the deployed system will be made available to the public.

At the heart of any internet voting system is a cryptographic protocol. The security of this protocol is a necessary requirement for trust in the internet voting solution. As part of the initial tender evaluation, a preliminary analysis of the cryptographic voting protocols was carried out. After the winning bid was selected, it was decided to make some modest changes to the cryptographic protocol, partially to get security proofs. This paper is a first analysis, focusing on the modified parts of the cryptographic protocol.

Internet voting in Norway Norwegian elections are somewhat complicated, but ballots essentially consists of a short sequence of options (a party list followed by selection of

candidates, at most about a hundred options in total) chosen from a small set of possible options (at most a few thousand). Note that the entire sequence is required to properly interpret and count the vote. For parliamentary elections order within the sequence is important, while order does not matter for county and municipal elections.

A real-world internet voting system has significant functional constraints.

The voter should not have to interact with the voting system more than once to submit a ballot. Most ballots will be submitted during peak hours, and the submitted ballots must be processed quickly. Once the ballot box closes, the result must be available as soon as possible.

Since cost does matter and secure computing hardware is expensive, any election infrastructure will have quite limited computational resources available for the protocol execution.

We also get functional constraints from security considerations. In practice, the two most significant security problems with internet voting will be compromised computers and coercion.

Since a significant fraction of home computers are compromised, the protocol must allow voters to detect ballot tampering without relying on computers. This is complicated by the fact that voters are unable to do even the simplest cryptographic processing without computer assistance.

When voting from home, no amount of cryptography can protect the voter from coercion. To defend against coercion, we mandate that the system must allow voters to vote multiple times, so-called revoting, counting only the final ballot. Voters may also vote once on paper and this vote should be counted instead of any electronic ballot, no matter when submitted. The internet voting system must essentially allow election administrators to cancel votes.

Defending against coercion by election insiders is very difficult. Before anyone can cast their vote, they must somehow authenticate to the system. For most plausible authentication systems, anyone with access to the authentication system will be able to detect electronic revoting.

In the Norwegian electoral system, for any ballot there will be a large number of legal ballots that are different, but have essentially the same effect on the final election result. Therefore, any coercer with access to the counted ballots (electronic or on paper) can tell his victim to submit an unlikely ballot with the desired effect, then verify that his victim did not revote by observing if the unlikely ballot is preset among the counted ballots.

Related work We can roughly divide the literature into protocols suitable for voting booths [5, 6, 16, 17, 18, 19], and protocols suitable for remote internet voting [7, 8, 14], although protocols often share certain building blocks. One difference is that protocols for voting booths should be both coercion-resistant and voter verifiable, while realistic attack models (the attacker knows everything the voter knows) for remote internet voting probably make it impossible to achieve both true voter verifiability and coercion-resistance.

For internet voting protocols, we can again roughly divide the literature into two main strands distinguished by the counting method. One is based on homomorphic tallying. Ballots are encrypted using a homomorphic cryptosystem, the product of all the ciphertexts is decrypted (usually using some form of threshold decryption) to reveal the sum of the ballots. For simple elections, this can be quite efficient, but for the Norwegian elections, this quickly becomes unwieldy, if not impossible.

The other strand has its origins in mix nets [3]. Encrypted ballots are sent through a mix net. The mix net ensures that the mix net output cannot be correlated with the mix net input. There are many types of mixes, based on nested encryption [3] or reencryption, verifiable shuffles [12, 17] or probabilistic verification [1, 14], etc. These will be quite efficient, even for the Norwegian elections.

Much of the literature ignores the fact that a voter simply will not do any computations. Instead, the voter delegates computations to a computer. Unfortunately, a voter's computer can be compromised, and once compromised may modify the ballot before submission.

One approach is so-called preencrypted ballots and receipt codes [4, 2], where the voter well in advance of the election receives a table with candidate names, identification numbers and receipt codes. The voter inputs a candidate identification number to vote and receives a response. The voter can verify that his vote was correctly received by checking the response against the printed receipt codes. Note that unless such systems are carefully designed, privacy will be lost.

One natural approach for securely generating the receipt codes is to use a proxy oblivious transfer scheme [13]. A ballot box has a database of receipt codes and the voter's computer obliviously transfers the correct one to a messenger, who then sends the receipt to the voter. The receipt codes should be sent through an independent channel. This ensures that a voter is notified whenever a vote is recorded, preventing a compromised computer from undetectably submitting ballots on the voter's behalf.

Our contribution The cryptographic protocol to be used in Norway is designed by Scytl, a Spanish company. It is mostly a fairly standard internet voting system. Essentially, a voter uses his computers to submit a ballot to an election infrastructure. To defend against coercion, a voter is allowed to submit multiple ballots, where the final submission will be counted. The *voter* gives his ballot to a *computer*, which encrypts the ballot and submits it to a *ballot box*. Once the ballot box closes, the submitted ciphertexts are decrypted in some *decryption service*, based on a reencrypting mix net. An *auditor* supervises the entire process.

The non-standard part of the system is detecting when a compromised computer has altered the ballot. The ballot box and a *receipt generator* cooperate to compute a sequence of receipt codes for the submitted ballot. These codes are sent to the voter through an independent channel. The voter verifies the receipt codes against a list of precomputed receipt codes printed on his voting card.

Scytl originally proposed to use a pseudo-random function family to compute the receipt

codes. While this would most likely be secure, it was difficult to prove anything about the proposal. It was therefore decided to use a slightly different construction. We use the fact that exponentiation is in some sense a pseudo-random function [9, 11], and since ElGamal is homomorphic, exponentiation can be efficiently done "inside" the ciphertext. This gives us a novel and quite efficient scheme.

We note that the protocol described in this paper is a simplified version of the protocol that will be deployed. The essential difference is that the simplified protocol is only secure against passive corruption of infrastructure players. The full protocol has protection against active corruption of infrastructure players. A description and analysis of the full protocol will appear later.

Overview of the paper We describe the security goals for the full protocol in Section 2, as we believe this is of independent interest. We discuss what capabilities an adversary will have, and define what the protocol is supposed to achieve.

The simplified protocol is specified and analysed in Section 3. We also briefly discuss how the full protocol protects against active attacks.

2 Security goal

Functionally, our protocol must allow voters to submit, repeatedly, ballots consisting of a sequence of options, each chosen from a set of options $\mathcal{O}$, to an infrastructure. After the voting period, the submitted ballots should be counted.

To define security for a protocol, we must define what kind of attackers we face and how we will allow them to influence the election.

We stress that we do not consider coercion in this analysis, beyond the brief discussion in the introduction.

The attacker We start with the standard premise that the attacker controls the network. What remains to decide is which players can be corrupted.

Any external attacker will clearly be able to compromise a number of voters as well as a larger number of computers.

If an infrastructure is divided into a small number of separate players, organizational and non-cryptographic technical measures may make it reasonable to assume that an inside attacker can compromise at most one infrastructure player.

Remark 1. Suppose we have some protocol satisfying the following: (i) the voter submits his vote to the computer, the computer submits an encrypted ballot to the infrastructure, and the infrastructure players cooperate to generate a receipt code and send it directly to the voter; (ii) a single infrastructure player X is responsible for sending the receipt code to the voter.

Consider the following attack where X and the voter's computer are corrupt. The computer first submits the voter's real ballot to the infrastructure, but the corrupted X delays the receipt code to the voter. Next, the computer submits a forged ballot to the infrastructure, leading to the computation of a new receipt code. This code is discarded by X, who instead sends the previous receipt code to the voter. The voter now believes his ballot was received correctly. But in reality, the voter's ballot has been replaced by a forged ballot.

The conclusion is that it is impossible for this type of protocol to protect a voter if both his computer and one particular infrastructure player is corrupt. For practical reasons, our protocol will be of this form. We therefore arrive at the following static corruption model:

> The attacker may corrupt either (i) any single infrastructure player and any subset of voters and computers, or (ii) the receipt generator.

For the simplified protocol, we use a weaker attack model:

> The attacker may corrupt either (i) any subset of voters and computers, or (ii) passively any one infrastructure player.

The attacker's influence We shall allow corrupt voters to submit spoilt ballots, that is, they can submit ballots containing a sequence of options from a set $\mathcal{O}'$ containing $\mathcal{O}$ as a subset.

Since an attacker controls the network between a voter's computer and the election infrastructure, he will certainly be able to delay or block the submission of ballots. Obviously, any corrupt infrastructure player can halt and thereby stop the election. They can usually cause more suspicious integrity failures, again stopping the election.

For usability reasons, the number of receipt codes must be equal to the number of options chosen. Therefore, if the receipt generator is corrupted, it is unavoidable that the number of options on a ballot leaks.

If the voter's computer is compromised, the attacker will see the ballot. The attacker may also modify the ballot, but in this case, the voter should be able to notice with high probability.

Remark 2. Any function from $\mathcal{O}'$ to the set of receipt codes defines a partition of $\mathcal{O}'$. A partition defines an equivalence relation. The uniform distribution on the set of functions from $\mathcal{O}'$ to the receipt code set therefore induces a probability distribution on the set of equivalence relations on $\mathcal{O}'$.

We extend an equivalence relation $\sim$ on $\mathcal{O}'$ in the obvious way to strings of options, i.e. $(v_1, \ldots, v_k) \sim (v'_1, \ldots, v'_{k'})$ if and only if $k = k'$ and $v_i \sim v'_i$ for $i = 1, \ldots, k$.

The probability that changing a ballot escapes detection should be equal to the probability that the original and modified ballots are equivalent under an equivalence relation sampled from the above distribution.

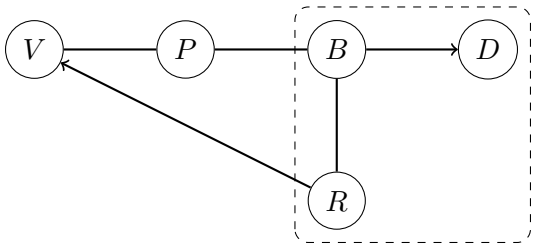


Figure 1: Communication between players in the simplified protocol. The infrastructure players are inside the box.

3 Protocol

We describe the simplified protocol. The players in the protocol are the *voter* V, the voter's *computer* P, the *ballot box* B the *receipt generator* R, and the *decryption service* D. The *auditor* is not part of the simplified protocol. The players communicate via secure, authenticated channels, as described in Fig. 1. We note that the ballot box knows which voter voter is communicating with which computer.

The voter chooses as its ballot a sequence of *options* $(v_1, \ldots, v_k)$ from a *set of options* $\mathcal{O} = \{1, 2, \ldots\}$, the computer pads the ballots with zeros to a fixed length k_{max}, encrypts it with the election encryption key and submits the encrypted ballot to a ballot box. The ballot box, in cooperation with the receipt generator computes a *receipt code* that is sent directly to the voter. The voter has a correspondence between options and receipt codes. If the receipt code received matches the option selected, the voter accepts, otherwise he knows something went wrong.

When the ballot box closes, the ballot box submits the encrypted ballots to the decryption service, which decrypts the ballots and publishes the result.

Prerequisites The system uses a finite cyclic group G of prime order q generated by g. We also have a pseudo-random function family F from G to $\mathcal{C}$.

A function $f : \mathcal{O} \to G$ is chosen. We then extend the function by defining $f(0)$ to be the identity element in G.

Key generation Before the election, three secret parameters a_1, a_2 and a_3 are generated such that $a_1 + a_2 \equiv a_3 \pmod{q}$. The ballot box gets a_2, the receipt code generator gets a_3 and the decryption service gets a_1. Three public parameters for the election, y_1, y_2 and y_3, are computed as $y_1 = g^{a_1}$, $y_2 = g^{a_2}$ and $y_3 = g^{a_3}$.

For every voter, s is sampled from $\{0, 1, \ldots, q-1\}$, and d from F. The composition of f, the exponentiation map $x \mapsto x^s$ and d gives a function $r : \mathcal{O} \to \mathcal{C}$ for each user, $r(v) = d((f(v))^s)$. Before the election, the set $\{(v, r(v)) \mid v \in \mathcal{O}\}$ is computed and given to V.

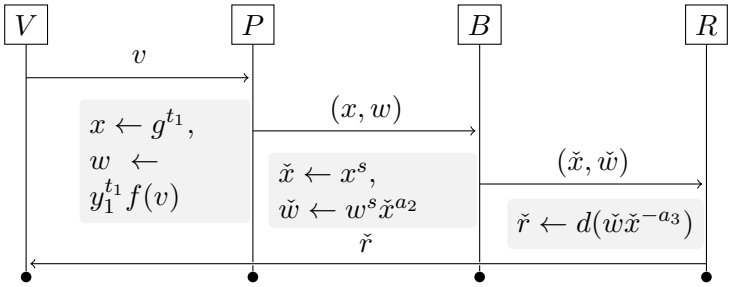


Figure 2: Protocol for submission of one option and generation of one receipt code.

Vote submission When the voter V wants to submit the ballot $(v_1, \ldots, v_k)$, the protocol proceeds as follows:

1. The voter sends $(v_1, \ldots, v_k)$ to his computer P. The computer sets $v_i = 0$, $i = k+1, \ldots, k_{max}$.
2. For $1 \leq i \leq k_{max}$, the computer samples $t_{1,i}$ from $\{0, 1, \ldots, q-1\}$, computes $(x_i, w_i) = (g^{t_{1,i}}, y_1^{t_{1,i}} f(v_i))$ and sends $((x_1, w_1), \ldots, (x_{k_{max}}, w_{k_{max}})$ to the ballot box B.
3. The ballot box computes $\check{x}_i = x_i^s$ and $\check{w}_i = w_i^s \check{x}_i^{a_2}$. The pairs $((\check{x}_1, \check{w}_1), \ldots, (\check{x}_{k_{max}}, \check{w}_{k_{max}}))$ and the voter's name is sent to R.
4. The receipt generator computes $\check{r}_i = d(\check{w}_i \check{x}_i^{-a_3})$ and sends $(\check{r}_1, \ldots, \check{r}_k)$ to the voter. (Note that k can be deduced from the number of non-identity decryptions.)
5. The voter verifies that every pair $(v_i, \check{r}_i)$ is in the set of receipt codes received before the election, and if so consideres the ballot cast.

The protocol is summarized in Fig. 2.

Counting When the ballot box closes, it sends every voter's final submitted encrypted ballot to the decryption service, in random order. The decryption service decrypts all the ciphertexts and publishes the resulting ballots in random order.

3.1 Completeness

The protocol is *complete* if, when every participant is honest, the submitted ballots are eventually correctly decrypted, and the receipt codes sent to the voter matches the expected values.

Completeness is obvious, except for the receipt code received by the voter. We only need to argue that $(v, \check{r})$ will always be in the computed set of receipt codes, that is, that

$\check{r} = r(v)$. We compute

$$\check{w}\check{x}^{-a_3} = w^s \check{x}^{a_2}\check{x}^{-a_3} = w^s\check{x}^{-a_1} = w^s(x^s)^{-a_1} = (wx^{-a_1})^s = (f(v))^s.$$

Completeness follows.

3.2 Security

This section argues informally about the security of the simplified protocol. We consider the following corruption model: (a) The voter and his computer are corrupted; (b) the voter's computer is corrupted; and (c) one of the infrastructure players is honest, but curious. We prove the following properties:

1. If a corrupt computer modifies a ballot, the voter will most likely discover this.
2. No honest, but curious infrastructure player will learn any non-trivial information about the ballots.

(a) By the assumption of authenticated channels, the ballot box can trivially ensure that at most one ballot is counted per voter. Submitting malformed ciphertexts will at most invalidate the voter's ballot, which we expressly permit.

(b) Suppose the computer submits $(v'_1, \ldots, v'_{k'})$ instead of $(v_1, \ldots, v_k)$.

We know that the exponentiation map is a permutation on G and that f is an injection. Since d will look like a random function, f composed with a permutation composed with a random-looking function will again look like a random function from $\mathcal{O}'$ to $\mathcal{C}$.

By Remark 2, the voter will accept the manipulation if and only if $(v'_1, \ldots, v'_{k'}) \sim (v_1, \ldots, v_k)$. As long as the set $\mathcal{C}$ is sufficiently large, this probability will be small.

(c) We consider the three infrastructure players in turn. Since they are honest, but curious, we only need to simulate the input they would normally see, we do not need to model interaction with other parts of the system.

The ballot box Suppose we have an honest, but curious ballot box B^* that after the election is over looks at the ciphertexts and outputs some information about the ballots submitted.

Given a tuple (g, y_1, u_1, u_2) of elements from G, we shall employ B^* as follows:

1. Generate a_2 and compute $y_3 = y_1 g^{a_2}$. Send a_2 to B^*.
2. Instead of encrypting the encoded option $f(v_i)$ as usual, we compute the encryption as $(x_i, w_i) = (g^{t_{1,i}} u_1^{t'_i}, y_1^{t_{1,i}} u_2^{t'_i} f(v_i))$, for some random t'_i.

If (g, y_1, u_1, u_2) is a Diffie-Hellman tuple, this will simulate the ballot box input perfectly. If (g, y_1, u_1, u_2) is not a Diffie-Hellman tuple, the ballot box input will contain no information at all about the ballots.

The conclusion is that if B^* can extract some information about the ballots, we have a distinguisher for the Decision Diffie-Hellman problem.

The receipt generator We shall now assume that the family of functions from $\mathcal{O}$ to G given by $v \mapsto f(v)^s$ is a pseudo-random function family, that is, functions sampled uniformly at random from the family are indistinguishable from functions sampled uniformly at random from the set of all possible functions from $\mathcal{O}$ to G. We shall discuss this assumption and the choice of function $f : \mathcal{O} \to G$, as well as an alternative assumption, in Sect. 3.3.

Suppose we have an honest, but curious receipt generator R^* that after the election is over outputs some non-trivial information about the ballots submitted.

Given a function f and a function $\rho : \mathcal{O} \to G$, we use R^* as follows:

1. For the voters $V_1, V_2, \ldots V_{j-1}$, we choose random functions $\rho_l : \mathcal{O} \to G$, $1 \leq l < j$.
2. Generate keys as usual for $V_{j+1}, \ldots, V_N$ and use the functions $\rho_l : v \mapsto f(v)^{s_l}$, $j < l \leq N$.
3. For V_j, we use the given function ρ.
4. For every ballot $(v_1, \ldots, v_k)$ from a voter V_l with function ρ_l, we compute $(\check{x}_i, \check{w}_i)$ as $(g^{t'_i}, y_3^{t'_i} \rho_l(v_i))$.

If our function ρ comes from the family and $j = 1$, this will simulate the receipt generator input perfectly. If our function ρ is a random function and $j = k_{max}$, the receipt generator input will contain no non-trivial information about the ballots submitted.

After a standard hybrid argument, the conclusion is that if R^* can extract some non-trivial information about the ballots, we have a distinguisher for the function family.

The decryption service Since the decryption service sees the encrypted ballots in random order, it does not know which ballot originated with which voter, hence can extract no information about which ballot belongs to which voter.

3.3 Encoding Options as Group Elements

The Decision Diffie-Hellman problem can be formulated as follows:

> Given $(g_1, g_2) \in G \times G$ (where at least g_2 is sampled at random), decide if $(x_1, x_2) \in G \times G$ was sampled uniformly from the powers of (g_1, g_2) (i.e. $(x_1, x_2) = (g_1^s, g_2^s)$ for some s), or uniformly from $G \times G$.

It is well-known (e.g. [9, 11]) that this is equivalent to the following problem:

> Given $(g_1, \ldots, g_n) \in G^n$ (where at least $g_2, \ldots, g_n$ are sampled at random), decide if $(x_1, \ldots, x_n) \in G^n$ was sampled uniformly from the powers of $(g_1, \ldots, g_n)$ (i.e. $(x_1, \ldots, x_n) = (g_1^s, \ldots, g_n^s)$ for some s), or uniformly from G^n.

Therefore, if we choose a random injection $\mathcal{O} \to G$ as our encoding function f, the assumption used to prove privacy against the receipt generator in the previous section follows easily from Decision Diffie-Hellman.

However, choosing a different encoding function will allow a significant (by a factor of 20-100) speedup of vote decryption. Let p be a safe prime, let G be the quadratic residues in $\mathbb{F}_p^*$ and let L be a set of small primes whose images $\{\ell_1, \ell_2, \ldots, \ell_n\}$ in $\mathbb{F}_p^*$ are quadratic residues. Consider the following problem:

> Given $(\ell_1, \ldots, \ell_n) \in G^n$ as above, decide if $(x_1, \ldots, x_n) \in G^n$ was sampled uniformly from the powers of $(\ell_1, \ldots, \ell_n)$ (i.e. $(x_1, \ldots, x_n) = (\ell_1^s, \ldots, \ell_n^s)$ for some s), or uniformly from G^n.

Now, if f is any injection from $\mathcal{O}$ into $\{\ell_1, \ldots, \ell_n\}$, the assumption used to prove privacy against the receipt generator in the previous section holds if the above problem is hard.

While this assumption is very similar to Decision Diffie-Hellman, it seems unlikely that it will be possible to prove that it follows from Decision Diffie-Hellman.

We currently believe that the best way to solve Decision Diffie-Hellman is to compute one of the corresponding discrete logarithms. It is known [15] that solving the static Diffie-Hellman problem with a fairly large number of oracle queries is easier than solving the discrete logarithm problem.

For fairly large n, a static Diffie-Hellman solver could be applied to decide the above problem. This would be faster than the fastest known solver for the Decision Diffie-Hellman problem in the same group. However, for our application, n will always be small, hence the static Diffie-Hellman solver can not be applied. It seems as if the best approach to solving the above decision problem is computing discrete logarithms.

Other approaches While the assumption discussed above is sufficient for security, it is not necessary. A weaker, sufficient condition would be if, given a permutation of a subset of $\{\ell_1^s, \ldots, \ell_n^s\}$ for some random s, it was hard to deduce any information about the which primes were involved and what the permutation was.

The simplest case, which happily corresponds to the most common voting pattern, is that the receipt generator sees one group element, and must decide which prime was used to generate it. For reasons explained in Sect. 3.4, the receipt generator is also given a random generator g and g^s.

We do the calculations for the case when there are only two primes to decide between, say ℓ_0 and ℓ_1. Let R^* be an algorithm that takes as input five group elements and outputs

0 or 1. Define

$$\begin{aligned} p_{00} &= \Pr[R^*(\ell_0, \ell_1, g, g^s, \ell_0^s) = 0], \\ p_{11} &= \Pr[R^*(\ell_0, \ell_1, g, g^s, \ell_1^s) = 1], \text{ and} \\ p_{i,\mathrm{rnd}} &= \Pr[R^*(\ell_0, \ell_1, g, g^s, g^t) = i],\ i \in \{0, 1\}, \end{aligned}$$

where s and t are sampled uniformly at random from $\{0, 1, \ldots, q-1\}$. Note that $p_{0,\mathrm{rnd}} = 1 - p_{1,\mathrm{rnd}}$, since the input distribution to R^* is identical for both probabilities.

We may define the advantage of R^* as $|p_{00} + p_{11} - 1|$. Observe that if $|p_{00} - p_{0,\mathrm{rnd}}|$ or $|p_{11} - p_{1,\mathrm{rnd}}|$ are large, we have a trivial solver for Decision Diffie-Hellman with the generator fixed to either ℓ_0 or ℓ_1.

We may assume that $p_{00} + p_{11} - 1 = 2\epsilon > 0$. Then either $p_{00} \geq 1/2 + \epsilon$ or $p_{11} \geq 1/2 + \epsilon$, so assume the former. Furthermore, let $p_{00} - p_{0,\mathrm{rnd}} = \mu$. If $|\mu| \geq \epsilon$, we have an adversary against Decision Diffie-Hellman with the generator fixed to ℓ_0, so assume $|\mu| < \epsilon$. Then

$$p_{11} - p_{1,\mathrm{rnd}} = 1 + 2\epsilon - p_{00} - (1 - p_{0,\mathrm{rnd}}) = 2\epsilon - \mu \geq \epsilon,$$

which means that we must have an adversary with advantage ϵ against Decision Diffie-Hellman with the generator fixed to either ℓ_0 or ℓ_1.

The same arguments applies to an R^* that can decide between multiple primes, he must lead to a successful adversary against Decision Diffie-Hellman with the generator fixed to one of the primes.

Unfortunately, the above argument breaks down if R^* is allowed to see multiple primes raised to the same power. It is possible that a more careful analysis could succeed here, though.

3.4 Active attacks

In this section, we briefly discuss how the full protocol will defend against active attacks from infrastructure insiders. The basic idea is that everyone should prove that they have faithfully executed the protocol. The computer proves knowledge of ciphertext content and the ballot box proves the correctness of its computations. In addition, the voter in cooperation with the computer signs the submitted ballot with a digital signature from a PKI, and the receipt generator signs a hash of the ballot which is then given to the voter as a second receipt.

We discuss each player in some detail.

The voter The voter must, in cooperation with the computer, sign the submitted ballot. Intuitively, this prevents a corrupt ballot box from inserting forged ballots, or falsely claiming that a given ballot belongs to someone else. With signed ballots, it is trivial to ensure that at most one ballot is counted per voter.

The computer The computer should prove that it knows the content of every submitted ciphertext (x, w). Intuitively, this is to prevent a corrupt ballot box from using a corrupt voter to submit honest voter's ciphertexts as its own, then learn the ballot contents from the receipt codes. Proof-theoretically, we will need to extract the vote from the ballot.

In theory, the proof of knowledge should have a proper online extractor [10], but morally a more efficient non-interactive Schnorr proof will do, even if we do not have a knowledge extractor for parallell composition.

The ballot box The ballot box must show the receipt generator the signed ballot, and also prove that $(\check{x}_i, \check{w}_i)$ is computed correctly. Intuitively, this is to prevent a corrupt ballot box cooperating with a corrupt voter from misusing the receipt generator's decryption capability.

In order to verify correct computation, the receipt generator must now see the entire ballot, including the voter's signature and the computer's proofs of knowledge. To simplify the security proof, the ballot box will also randomize the ciphertext $(\check{x}_i, \check{w}_i)$, although we believe a somewhat more involved proof will allow us to dispense with the randomization.

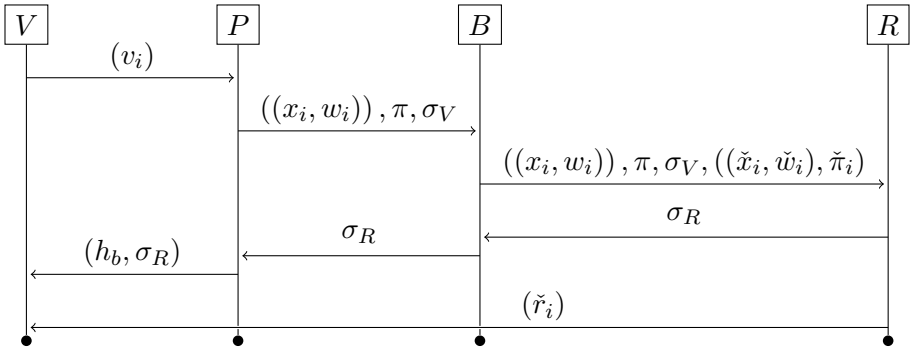
The ballot box computes:

$$\begin{aligned}(\bar{x}, \bar{w}) &= (x^s, w^s),\\ \hat{w} &= \bar{x}^{a_2},\\ (\tilde{x}, \tilde{w}) &= (g^{t_2}, y_3^{t_2}),\\ (\check{x}, \check{w}) &= (\bar{x}\tilde{x}, \bar{w}\hat{w}\tilde{w}).\end{aligned}$$

We need to hide $\hat{w}$ and $\tilde{w}$. For this, we use ElGamal encryption with a random public key as a commitment scheme. We can then use the fact that ElGamal commitments are homomorphic, computationally hiding and unconditionally binding to create a reasonably efficient proof of correct computation, essentially based on a Schnorr proof of knowledge of a discrete logarithm. Interestingly, for this proof we do not need an online extractor.

Receipt generator The receipt generator verifies the voter's signature and every proof, then signs a hash of the ballot and returns the signature to the ballot box (who should forward the signature to the voter's computer). Without this signature, the voter's computer will not inform the user that the ballot has been accepted. If the ballot box discards a ballot, this signature will allow a voter to prove, in cooperation with the auditor, that his ballot was discarded.

We summarize the submission of a ballot and generation of receipt codes in the following diagram. Here, π and $\check{\pi}_i$ are non-interactive proofs, h_b is a hash of the encrypted ballot, and σ_V and σ_R are signatures.



Decryption service The decryption service is a reasonably standard system consisting of a mix net followed by verifiable decryption. The mix net is similar to randomized partial checking [14] or "almost entirely correct mixing" [1]. We defer the description and analysis to the full version of the paper.

Auditor The auditor receives the entire content of the ballot box and a list of hashes of encrypted ballots seen by the receipt generator. The auditor verifies the content of the ballot box (signatures and proofs), that no ballots have been inserted or lost compared to the receipt generator list and computes on its own a list of encrypted ballots that should be counted. The auditor compares this list to the ciphertexts input to the mix net, then verifies the proofs offered by the mix net and the decryption service. The auditor also publishes hashes of every ballot, so that voters can verify that their ballots were included in the count.

4 Conclusion

We have described a simplified version of the cryptographic protocol that will be used in the Norwegian goverment's e-voting experiment in 2011 and analysed its security. While this method relies on a new and untested cryptographic assumption for security from the receipt generator, we believe this is justified for the following reasons:

- The receipt generator will be quite well protected, and a compromise of this system is quite unlikely.
- Most voters submit ballots with exactly one option. For such ballots, the problem facing the receipt generator is essentially equivalent to Decision Diffie-Hellman with a fixed generator, and it would be quite surprising if any such problem was easy.
- While the efficiency gains for mixing and decryption are significant, it is possible to sacrifice these gains to get a less efficient system whose security is based entirely on Decision Diffie-Hellman.

Future work There are several ways in which the full protocol or its analysis can be extended. One possible idea is to have the voter's computer generate ciphertexts $(\hat{x}_i, \hat{w}_i)$ and prove that they have the same decryptions as the ciphertexts (x_i, w_i). Then the ballot box computes $(\check{x}, \check{w})$ directly, obviating the need for the secret key y_2 and the relationship $y_1 + y_2 \equiv y_3$. To prevent the receipt generator from simply decrypting $(\hat{x}_i, \hat{w}_i)$, the voter's computer must actually submit a commitment to $\hat{w}_i$ as well as an opening that is only seen by the ballot box. Every proof must then work with the commitment, most likely increasing total computational load.

If both the ballot box and the receipt generator are compromised and cooperate, election privacy will still be lost even if the election decryption key is not revealed. The reason is of course that the ballot box and the receipt generator together could decrypt $(\hat{x}_i, \hat{w}_i)$. However, if the system is only partially compromised (say, the keys and the content of the ballot box leak after the election), it is possible to preserve election privacy if $(\hat{x}_i, \hat{w}_i)$ and $(\check{x}_i, \check{w}_i)$ are discarded after use. It is unclear if the modest increase in robustness justifies the cost.

If the public key infrastructure in use provides each voter with a public key encryption functionality, in addition to the identification and signature functionality, it might be possible to move the entire computation of $(\check{x}, \check{w})$ to the voter's computer. This should be safe because the receipt generator applies a secret, pseudo-random function to the decryption to get the receipt codes. In such a scheme, the ballot box might not need any secret keys at all, a significant system simplification.

References

[1] Dan Boneh and Philippe Golle. Almost entirely correct mixing with applications to voting. In Vijayalakshmi Atluri, editor, *ACM Conference on Computer and Communications Security*, pages 68–77. ACM, 2002.

[2] e-voting security study. CESG, United Kingdom, July 2002. Issue 1.2.

[3] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[4] David Chaum. Surevote. `www.surevote.com`, 2000.

[5] David Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy*, 2(1):38–47, 2004.

[6] David Chaum, Peter Y. A. Ryan, and Steve A. Schneider. A practical voter-verifiable election scheme. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *ESORICS*, volume 3679 of *Lecture Notes in Computer Science*, pages 118–139. Springer, 2005.

[7] Josh D. Cohen and Michael J. Fischer. A robust and verifiable cryptographically secure election scheme (extended abstract). In *Proceedings of 26th Symposium on Foundations of Computer Science*, pages 372–382. IEEE, 1985.

[8] Ronald Cramer, Matthew K. Franklin, Berry Schoenmakers, and Moti Yung. Multi-autority secret-ballot elections with linear work. In Ueli M. Maurer, editor, *EUROCRYPT*, volume 1070 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 1996.

[9] Ivan Damgård, Kasper Dupont, and Michael Østergaard Pedersen. Unclonable group identification. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 555–572. Springer, 2006.

[10] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 152–168. Springer, 2005.

[11] Kristian Gjøsteen. A latency-free election scheme. In Tal Malkin, editor, *CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 425–436. Springer, 2008.

[12] Jens Groth. A verifiable secret shuffle of homomorphic encryptions. In Yvo Desmedt, editor, *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 145–160. Springer, 2003.

[13] Sven Heiberg, Helger Lipmaa, and Filip van Laenen. On achieving e-vote integrity in the presence of malicious trojans. Submission to the Norwegian e-Vote 2011 tender, August 2009.

[14] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In Dan Boneh, editor, *USENIX Security Symposium*, pages 339–353. USENIX, 2002.

[15] Antoine Joux, Reynald Lercier, David Naccache, and Emmanuel Thomé. Oracle-assisted static diffie-hellman is easier than discrete logarithms. In Matthew G. Parker, editor, *IMA Int. Conf.*, volume 5921 of *Lecture Notes in Computer Science*, pages 351–367. Springer, 2009.

[16] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. Cryptology ePrint Archive, Report 2002/165, 2002. `http://eprint.iacr.org/`.

[17] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *ACM Conference on Computer and Communications Security*, pages 116–125, 2001.

[18] P. Y. A. Ryan and T. Peacock. Prêt à voter: a systems perspective. Technical Report CS-TR No 929, School of Computing Science, Newcastle University, September 2005.

[19] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *EUROCRYPT*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403. Springer, 1995.